第 1 章

コストとメリットを考えて構成管理を実践する
――最初は小さな一歩から始めよう

山崎辰雄，SESSAMEの仙人

構成管理は製品開発の基本であり，作ったものに不ぐあいが見つかった場合や作ったものを再利用する場合に必要となる．しかし，時間に追われて開発している場合，管理のための作業が負担になる．そこで，構成管理を実施するとどのようなメリットがあるのかを検討して，開発者の役に立つものだけを管理する．また，最初は小さいところから始めて，実績を積みながら適用範囲を広げていくとよいだろう．（編集部）

1 構成管理は製品開発の基本

私たちはさまざまな製品に取り囲まれ，それらを使いこなして生活しています．これらの中には，設計/製造時の問題からそのまま使い続けることができないと判定され，回収される製品があります[注1]．例えば，人命にかかわる製品の一つである自動車の市場回収についてインターネットで検索すると，2004年11月の1ヵ月間だけでも100万台が回収の対象とされていることがわかります．

不ぐあいの内容はというと，発火，発煙，走行不能，ハンドルが効かない，ブレーキが効かない，燃料供給不能，燃料もれ，ブレーキ・ランプが点灯しないなどさまざまで，読んでいると背筋が寒くなってきます．さらに，回収対象となるモデルの製作期間が「平成元年～平成12年」などといった長期にわたっているものもあります．

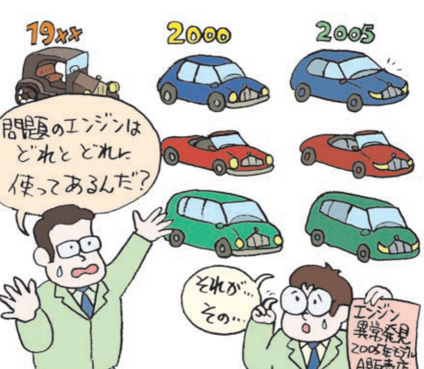


図1　管理できていないと全製品を回収するはめに…？

残念ながら，製品の構造がどんどん複雑になる中で，なんらかの不ぐあいが発生することはどのように努力しても避けられないことのようです．そこでこのような発表を行うことになるのですが，少なくとも次の情報を把握していなければ製品の回収に対応することができません．

- その不ぐあいはどのモデルで起こるのか
  （使われている技術や部品を把握していること）
- どの製作期間のものか
  （時期によって部品，技術が異なることがある）

さらに，直接顧客に呼びかけるためには，どの個体がどの顧客に渡っているのかという情報も必要です．

これらを整理すると，時間軸に沿って移り変わる「モデル（機種，品種）ごとの情報」と「個体ごとの情報」を管理すること，さらにそのような情報を（製品の寿命にもよるが）10年以上参照できる状態にしておかなければならないことがわかります（図1）．

● そのモデルは何から形作られているのか？

このうち「モデルごとの情報」とはどのようなものでしょうか？ 先ほど挙げた自動車を例に，モデルを形作る情報を思いつくまま列記してみましょう．

注1：製品回収情報については国民生活センターのWebサイト（URLは「http://www.kokusen.go.jp/recall/」）を，自動車のリコールの届け出については，国土交通省のWebサイト（URLは「http://www.mlit.go.jp/jidosha/recall/recall04/recall_.html」）を参照のこと．



図2　構成管理＝「もの」を構成する情報を管理すること

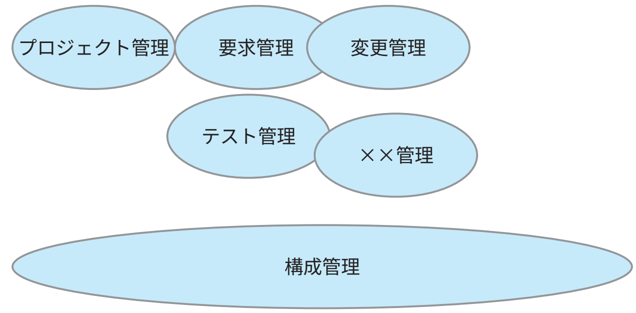


図3　構成管理はプロジェクト全体の管理の基盤である

構成管理は，製品を形作る構成要素と構成要素の間の関係を管理することを指す．なお，しばしば構成管理と対になって使用される「変更管理」は，構成要素ごとの変更に着目し，変更要求，変更の承認，変更の実施，変更の確認を着実に遂行できるように管理するものである．

- 目に見える「もの」にかかわる情報――自動車を形作る部品（「もの」そのもの），部品と部品の関係（それぞれの部品はどの部品から構成されているのか），自動車の取り扱い説明書や保証書（「もの」に付属する「もの」）
- 後で「もの」だけ見てもわからない情報――いつ作ったか，どうやって作ったか，だれが作ったか，どこから買ったか，どこに売ったか

これらのモデルを形作る（構成する）情報を構成情報と呼び，これらの情報を管理することを構成管理と呼びます（図2）．具体的には，個々の構成要素と構成要素の間の関係を管理します．また，時間軸上の構成要素やその関係の変化も管理します．なお，モデルに関する情報だけでなく，先ほど挙げたような「個体ごとの情報」を含め，「あるモデルにかかわるすべての情報」を直接，構成情報として取り扱ってもかまいません．このように，作るときだけでなく売った後についても情報を一元管理していれば，万が一市場から製品を回収するなどの緊急事態が発生したとしても，さまざまな対処が可能になります．

● 構成管理は管理の基盤となる

組み込み機器開発において，管理と名のつくものはいくつもあります（例えば，プロジェクト管理，要求管理，変更管理，テスト管理など）．構成管理はこれらの管理の基盤となります（図3）．

例えば，組み込み機器を形作る構成要素にはどのようなものがあるのでしょうか？ ハードウェアとしては，機器を構成する部品や回路図が挙げられます．ソフトウェアとしては，ソース・コード，バイナリ・データ，開発ツールのバージョン，テスト・パターン，実機テスト用の基板などが挙げられます．システムやハードウェア，ソフトウェアについて書かれた仕様書も管理の対象になります．プロジェクトの成果物に不ぐあいが発生したときに原因を解析し，修正版を作るためには（または，次のプロジェクトで環境や成果物を流用するためには），プロジェクトで利用した

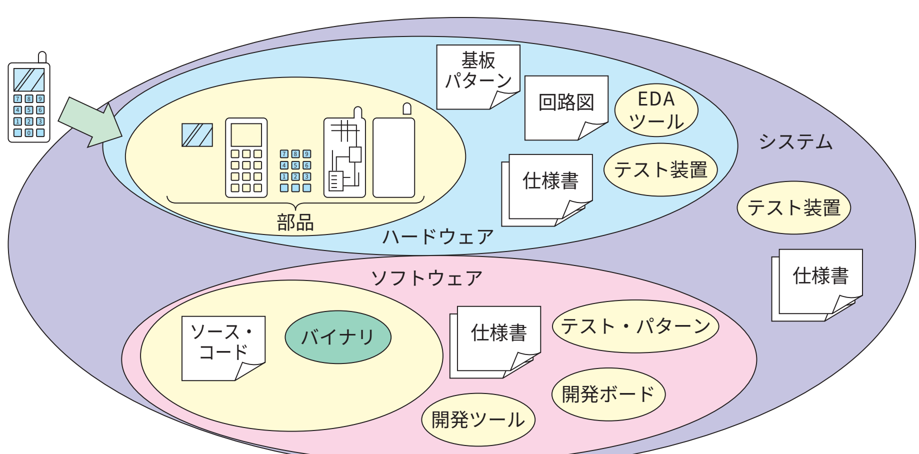


図4
組み込み機器の構成要素

このうちソフトウェア開発に関する部分が，ソフトウェア構成管理と呼ばれるものにあたる．ソフトウェア開発だけを考えても，構成管理範囲はいくらでも展開できる．開発プロジェクトや販売に携わる人，利用者まで含めることもできる．その範囲は際限がない．

ハードウェア，ソフトウェア，環境のすべてを管理しておく必要があります（**図4**）．

また，構成管理の概念をソフトウェア開発工程に絞って適用したものを，ソフトウェア構成管理（SCM：software configu-ration management）と呼びます．

● なぜ構成管理が必要なのか？

取り扱う「もの」が，何からどのように形作られているのかを知る方法がなければ，それを作り出したり，後から変更したりすることができません．ひとりでもの作りに励んでいるのであれば，その作業の間は「必要な知識はすべて頭の中に入っている」と言えるかもしれません．しかし，少し時間が経つと，知識はすぐ忘却の彼方に飛び去ってしまいます．

ましてや複数人で共同で作業を進めるときは，自分たちが作り出す製品がどのようなものになるのか，知識を効率的に伝え，共有し，お互いがどこの箇所を担当するのかを明示しなければなりません．構成管理はもの作りの基本なのです．

構成管理が必要となるプロジェクトの典型例を以下に挙げてみます．

1）一つの製品を分散した拠点で開発する場合

開発担当者が400人，テスト担当者が100人という体制で世界各地で分業して開発する組み込みシステムの開発があります．世界市場を相手にする製品では，それぞれの地域でもっとも得意とする技術を持ち寄って世界最高の製品の開発をめざすときにこのような体制がとられます（**図5**）．

このような開発を行うときには，だれがどの構成要素（部

Column

手近なシステムを構成管理してみよう

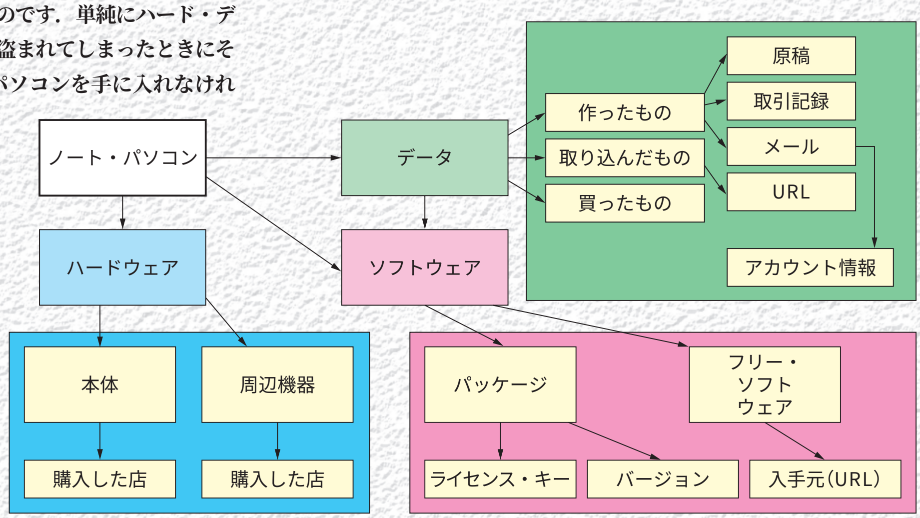
話だけ聞いていても，いま一つ理解しにくい，という方もいると思います．構成管理の実例として，手近な製品の構成管理を紹介します．構成管理というからには，複数の構成要素から成り立っていて，その構成要素の間の関係まで含めて管理が必要なものを対象としなければなりません．ちょうど，手もとにあるパソコンが手ごろな題材になりそうです．

そんなことを思いついたのも，筆者が今いちばん気がかりなことの一つが，ひたすら労力をつぎ込んで書いた原稿の入っているこのノート・パソコンの中身をどう守っていくかという問題を抱えているからです．ウィルスにやられてハード・ディスクを再フォーマットせざるをえなくなったらどうしよう？ 盗まれたら？ 電車の網棚に置き忘れたら？ パソコンをこっそり乗っ取る，たちの悪いソフトウェアを仕込まれてしまったら？ などなど，心配の種は尽きません．

「そんなの，バックアップをとっておけばいいじゃないか」と言いたいそこのあなた．事態はもう少し複雑なのです．単純にハード・ディスクのコピーを保管しておくだけでは，盗まれてしまったときにそっくり同じハードウェア構成のノート・パソコンを手に入れなければ，起動すらできない可能性があります．それに2年間も使っていると，ハード・ディスクには「こんなソフトウェアを入れるんじゃなかった」と後悔しているものまで入っています．残念ながら，現在のOS（とくにパソコン用のOS）では，アプリケーション・ソフトウェアをアンインストールしても，何らかの残がいがファイル・システムのディレクトリ構造のあちこちに残ってしまいがちです．こういうものはできるだけ取り除きたいのです．

となると，少なくとも次の要素を管理しなければならないことがわかります．

- 周辺装置（装置本体，デバイス・ドライバが入っているCD-ROM）
- アプリケーション・ソフトウェア
  （ソフトウェア本体が入っているCD-ROM，ライセンス・キー）
- ダウンロードしたお気に入りのフリー・ソフトウェア（URL）
- 執筆した原稿，やりとりしたメール



図A-1
パソコンの構成要素
ハードウェア（ノート・パソコンの本体や周辺機器）なら，購入した店の電話番号などは必要だろう．ソフトウェアはライセンス・キーを忘れるとインストールできなくなる．フリーウェアは入手したURLを記録しておくべきだろう．データも買ったもの，ダウンロードしてきたものなど，いろいろある．

品）を担当し，それがほかの構成要素とどのようにかかわり合い，どのタイミングで組み合わされるのかという情報の管理が非常に重要になります．

2）複雑な標準規格を実装する場合

ある通信家電製品の基本箇所の標準規格書の量は，今や一つの本箱では収まりません．しかも，四半期ごとに改訂が進められています（図6）．このように巨大で複雑な仕様書から着実にそれを実装した製品を作り出すことは，どれが何を参照して作られているのか，またそれをだれが担当しているのかという管理情報がなければ不可能です．また，それは製品の主要な仕様ではありますが，ほかにも従わなければならない規格は（文字どおり）山ほどあります．

ここで挙げた二つの例はどちらも規模の大きい開発の例



図5　分散開発の場合

そうそう，かんじんのパソコンの各種設定もあります．まだまだありそうです，リストにしなければまとめきれそうにありません．

ここで構成管理を行う目的をまとめておきます．すぐに思いつくのはこんなところでしょうか．

- トラブルに遭ってパソコンが使えなくなったときに，速やかに同じ環境を復旧したい
- 新しいパソコンを買ったときに，今までのパソコンと同じ環境にしあげたい

目的をはっきりさせたところで，管理対象となる構成要素を思いつくままに書き並べていきます（**図A-1**）．

これらの構成要素の中には，共通の要素もあります．例えば「製品」であれば，以下が共通要素となるでしょう．

- 製造元（会社名，連絡先，URL）
- 購入した店（会社名，連絡先，担当者）
- 製品のモデル名
- 製品のバージョン
- 製品のシリアル番号
- 製品のスペック（仕様）
- 解説書
- ほかに同梱されていたもの

そのほか，「設定」なら，設定内容，設定方法が共通要素となるでしょう．「作業のしかた」や「連絡先」，「場所」についてもそれぞれ共通要素があります．

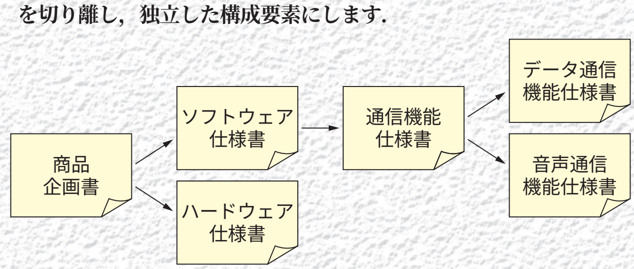
構成要素を挙げたら，次に構成要素の関係を考えます．例えばデータは，それを取り扱える（表示，編集できる）アプリケーション・ソフトウェアといっしょに取り扱う必要があります．つまり，データとソフトウェアの「関係」が管理対象になります．あるデータは，アプリケーション・ソフトウェアの特定のバージョンでないと開けないかもしれません．さらに，OSのバージョンとアプリケーション・ソフトウェアのバージョンの関係にも注意します．

加えて，運用に関する情報（最後にデータをバックアップしたのはいつか）も構成管理の対象となります．

● 構造を決定する

上記の構成要素を見る限り，住所録ツールを流用して製品情報や製造元を入力しておけば何とか管理できそうです．この例では，最低限の水準として，構成要素と構成要素間の関係を表現できればよいので，テキスト・エディタ（メモ帳）や表計算ソフトウェア（Excelなど），ハイパーテキスト（HTMLファイル）として保存したり，そのほかのデータベース・システムなどを利用してもよいでしょう．

なお，構成要素間の参照関係は一方向になるようにします（**図A-2**）．「複数の要素がある要素を参照する」，または「ある要素が複数の要素を参照する」という参照関係は問題ありません．記述が重複したり相互に参照していると，片方を更新したらもう片方も更新しなければならず，確認がやっかいになります（三つ以上の構成要素が循環参照していると，なお悪い）．このような場合は，相互に参照している部分を切り離し，独立した構成要素にします．



図A-2　構成要素間の参照関係は一方向になるようにする

木（ツリー）構造に表現できる参照関係が望ましい．木構造として表現できない関係（例えば，複数の文書に同じことが記述してあったり，ソフトウェアの仕様書とうたいながらハードウェアについても記述しているなど）は避けたほうがよい．

1



図6　標準規格を実装する場合

でしたが，構成管理は巨大で複雑なものを管理するためだけに使われるわけではありません．管理していないと手に余るという程度のもの（パソコンの構成など）を管理するのも「構成管理」です（pp.110-111のコラム「手近なシステムを構成管理してみよう」を参照）．たかがパソコンの構成ですが，きちんと管理しておくことにより，環境を再現したいときに（管理していなかった場合と比べて）かなり楽ができます．

● 理想的な構成管理を考える

目標を定めずにただ「管理する」と言っても，わざわざそのようなことをしたいとはだれも思いません．構成管理を行う前に，何が実現できたらうれしいのかを書き並べるところから始めてみましょう．

- 構成要素（何からできているか）がわかること
- 参照関係がわかること
- 変更の影響範囲がわかること
- 対応情報（何と何が対応しているか）がわかること
- 環境が壊れたときに復旧できること
- 新しい環境において，すべてを再構成できること

これらの項目をさらに具体的に考えてみます．

1）何からできているか──ハードウェアなら，筐体，プリント基板，半導体，回路図など．ソフトウェアなら，仕様書，ソース・コード，バイナリ・コードなど．

2）参照関係──「回路図を書くときに，データシートや仕様書を参照した」，「仕様書を書くときに，企画書とマニュアルを参照した」，「プログラムを作るときに，言語仕様書とソフトウェア仕様書を参照した」など．

3）変更の影響範囲──ある構成要素を変えたとき，影響がある範囲（ここも変えなければならない，という箇所）がわかれば，かなり楽ができます．わからない場合，影響を受けないかどうかを一つ一つ丹念に調べなければなりません．西暦年の管理を下2けたから4けたに変更したときの影響評価で大騒ぎしたこと（いわゆる「2000年問題」）は記憶に新しいところです．

4）対応情報──「このLSIに対応するソフトウェアのバージョンはこれ」，「この規格に対応しているチップセットはこれ」という，ある構成要素とある構成要素の間の対応情報が必要です．

5）環境の復旧，再構成──今や，一度作った構成要素は，ほかでもどんどん使い回すのが常識です．「これとこれは流用して，あとは取り除く」，「二つ足して新しいのを作る」などということを，あまり手間をかけずに実現したいものです．

そのほか，使用したソフトウェアの設定などといった環境構築情報も必要になるでしょう．これについては，例えば「ビデオで作業内容をすべて撮っておく」などという方法も思いつきますが，これでは後で必要な情報を取り出すときに非常に労力がかかることになります（コンピュータを使ってすべてを記録，検索できるようになれば，有効な方法かもしれない）．

● 構成管理導入の現実は厳しい

ところで，すでに構成管理を導入しているプロジェクトでは，どのようなことが起きているのでしょうか？ 当然ながら，うまくいっているプロジェクトと，うまくいかずに使うのをやめてしまったプロジェクトがあります．構成管理をうまく導入できなかったプロジェクトでは，どのようなことが起きたのでしょうか？

1）導入時：費用対効果の問題

つぎ込んだお金に見合うだけの成果が得られると納得できなければ，導入を見送ることになります．効果を測定する方法や，投資額に見合うかどうかを判定するための材料を見つけ出せるかどうかがかぎです．

2）運用時：ルール順守の問題

「このように使いましょう」と最初に決めても守れない，守らせることができないという問題が起こりがちです．例えば，時間がないからと構成管理情報を更新せずに現物（ソース・コードや仕様書など）のほうだけを更新してしまい，その結果，保持している構成管理情報が古いものになって，結局使わなくなってしまうという話をよく聞きます（図7）．

この問題は，構成管理にどのようなメリットがあるのか



図7　時間に追われ，構成管理情報を更新しなくなってしまいがち



図8　製品（構成要素）には賞味期限がある

という動機付けが不十分な場合と，トレーニングが不十分なために操作方法がわからず，そのまま使わなくなる場合があるようです．

3）機能が足りない

実際にツールを使って管理し始めたところ，機能に制約があって環境を丸ごと管理できなかったため，導入効果がなかったということも聞きます．ツールには得手不得手があります．何かを重視したためにほかの必要な機能がもれてしまい，せっかく導入したツールが活用されないことがあるようです．

4）管理対象が狭すぎる

管理対象とした領域（部門や組織）が狭かったため，実効性のある管理ができなかったという話を聞きます．これは，構成要素を管理する部門や組織が複数にわたっていて，統合的に管理できない場合に起こります．例えば，同じ製品に使われるソフトウェアでも，デバイス・ドライバを開発する部署と，アプリケーション・ソフトウェアを開発する部署が地理的に大きく離れていたり，決裁権を持つ管理者が異なっていたことにより，同じ作業方針に基づいて管理できない，などの場合があります．

5）環境を再現できない

構成管理を行って情報を記録していても，数年後に同じ構成要素をそろえて同じ環境を作れるとは限りません．構成要素を提供していたメーカが提供をやめてしまうことがあるからです（3年前に売られていたパッケージ・ソフトウェアやLSIを今でも購入できるかどうか，試してみたことはあるだろうか？）．

6）時間とともに価値が減ってしまう

市場で求められる技術や市場に提供される技術は，日々更新されています．しっかり管理して使える状態に維持していても，時間が経てば市場における価値を失ってしまうことがあります（図8）．「長い目で見て」投資していても，長い時間が経って古くなったものを維持するための投資は回収できなくなってしまいます．ものにも情報にも寿命があるのです．今後は構成要素の賞味期限も構成管理情報の一つとして定めておく必要があるでしょう．

7）使い回しには限界がある

ソース・コードや開発仕様書を再利用することはありますが，環境や担当者が変わると再利用できない可能性が高くなります．その結果，新しく一から開発したほうがよいという判断もあり得ます．

8）作っているときと保守するときの価値観は異なる

作っているときと再利用するときとで文書の重要性は変わります．作るときに重視されていた書類でも，再利用時にはまったく不要になるものがあるのです（例えば，スケジュール管理表や障害報告書など）．また，より重要性を増すものは仕様書と実装の差分情報です．

一般に，管理者が管理のために使用していた情報は不要となり，技術者が参照していた技術情報，仕様書と実装の対応，機能に関する情報（何があり何がないのか）などは再利用時にとても重要なものとなります．

このことを考えずに何を管理するのかを決めていると，再利用できない情報ばかりが保管されているという状態になってしまいます．

● 管理者のための管理にしない

それでは，どのように管理するのがもっとも良いのでしょうか？ 管理一般に対して「良い管理」を考えるとき，筆

者は以下の三つが重要であると考えます．

- もっとも負担の少ない方法で目的を達成できること
- 管理対象に「管理されている」と思わせずに，自然な形で管理できること．
- 管理対象がみずから管理すること（管理者はモニタリングに専念する）

管理作業の内容は，管理者ではなく技術担当者にとって必然性があるものにするべきです．管理のための管理作業にしてはなりません．作成された情報が何人に何回，どのくらいの期間利用されるのかを，作成のために費やされた時間と合わせて一覧表にして，むだな作業を削減します（**表1**）．これを実施すると，ものを作るのに「管理」のための情報は必須ではないことがよくわかります．本来の「もの作り」を最優先とする必要があります．管理はもの作りを支援する2次的なものにすぎないことを，管理者は肝に銘じるべきです．

同様に，構成管理も管理者のためのものではありません．もの作りに携わる人が作業しやすいように，資料や成果物をまとめるために実施するものです（**図9**）．手順としては，だれのための作業かを明確に定義することから始めます．そして，管理対象は作業者みずからが必要と認めるものだけに限定するとよいでしょう．



図9　構成管理の対象は作業者が必要と認めるものに限定する

構成管理の専任者を置くときには，あくまでも，もの作りに携わる人の補助に徹するように，その管理者の業務を定めます．作業者が希望するときは，その人に代わって構成を確認しながら管理できる環境を作ってもよいでしょう（これはレビューを兼ねた作業になる）．

構成管理のライフ・サイクル（開始，運用，管理情報の廃棄という一連の流れ）は，飛行機の運航に例えることができます．離陸（導入/体制確立），巡航（運営），着陸（終了/まとめ作業）からなり，離陸と着陸は困難ですが，巡航状態になればあまり手をかけなくてもかってに飛んでくれます．

●「次から始めよう」ではいつまでも実施できない

構成管理はいつから始めればよいのでしょうか？「次にプロジェクトが始まるまで導入を待とう」などと考えているとしたら，いつまでたっても導入することはできません．「次のプロジェクトが始まるまで待つ」ということは，今のプロジェクトには適用しないということになります．現在遂行しているプロジェクトは構成管理を行う必要がないのでしょうか？

また，こんな話をマネージャから聞いたことはありませんか？「次のプロジェクトは投資を有効活用するために，今のプロジェクトの成果物を再利用する」．成果物を再利用するためには，成果物を管理する必要があります．つまり，すでにある成果物を後から管理するわけです．何もないところから管理を始めるのは，きわめてまれなことなのです．

● 今より楽になることを作業者に理解させる

さて，すでにそこにあるものに対して管理を始めるためには，どのようにすればよいのでしょうか．管理するためには，管理対象となる作業者の人たちの協力が必要です．作業者は，ただでさえ忙しいところに余分な作業が追加さ

表1
作成された情報がどの程度利用されているのかの一覧表
この一覧表を見る限り，週報を書かせるのはやめたほうがよさそうだ．管理者が一人一人と直接話したほうが，より効率的に状況を把握できるだろう．

書類名	作るのにかかった時間	更新回数	見た人	何回参照したか	次機種で見るか
開発日程表	1週間	日程が変わるたび	関係者全員	数えきれず	計画するときには見る
週報(第1週)	30分	更新なし	管理者	1回	見ない
週報(第2週)	30分	更新なし	管理者	1回	見ない
週報(第n週)	30分	更新なし	管理者	1回	見ない
規格文書一覧	不明	数えきれず	開発者全員	数えきれず	必須
××仕様書	1週間	仕様変更のたび	プログラマ	数えきれず	必須

れるとしたら「積極的にはかかわりたくない」と考えるでしょう．少なくとも「少し余分な作業を行うことにはなるけれど，結果的にはその作業を行ったことによってほかの作業が大幅に減り，楽ができる」ことを示さない限り，協力は得られないでしょう．

実績を積んだうえでわかりやすく提示できれば，協力を得やすくなります．筆者としては，小さいところで始めてしだいに適用範囲を広げていくことをお勧めします．ある程度お金と時間をかけて納得させなければ，管理作業は組織内に定着・浸透しません．

今あるものを構成管理下におくためには，棚卸し（もしくはサルベージ）と呼ばれる作業が必要です．これは，構成要素として何があるのか，構成要素間の関係はどうなっているのかなどを調べる，いわゆる洗い出し作業です．この作業も結果的にプロジェクトを検証することになり，不ぐあいを先取りして検出できる可能性があります．レビューしながら結果をまとめるときにこの洗い出し作業を実施し，データベースを構築していくとよいでしょう．

手始めとしては，あくまでも今までの業務の延長線上で，「関係するものをまとめて整理する」という程度の意識で，実施できる範囲に留めるべきです．

過去に構成管理を行ったプロジェクトの経験があるのなら，プロジェクトの開始時点から全面的に導入することも可能でしょう．この場合，新しくプロジェクトに入った人がとまどわずにすむように，手順や考えかたを具体例とともに文書化し，十分なトレーニングを受けられるようにしておく必要があります．

● 構成管理を設計する

構成管理を行うには，構成要素をどの程度きめ細かく管理するのかという「管理の粒度」を定める必要があります．このとき，コストとメリットのバランスを考えることがたいせつです．当然ながら，構成要素の定義を細かくするほどコストがかかります．構成要素の数が増えるからです（**図10**）．

また，管理対象となる成果物の整理を各個人に依頼すると，まとめかたが多種多様なものになってしまいます．基本的な構造や事例を提示し，さらに個別に指導する必要があります．もちろん集合教育も可能ですが，それでは一時的な効果しか期待できません．しっかり身につけて実施してもらえる保証がないからです．1対1で，その人の成果物を対象として整理を進めていきます．横についていっしょに働くことで，管理の粒度を均一にし，有効な管理のしかたを身につけてもらうことができます．基本は構成要素と構成要素間の関係を見出すことにあります．

なお，構成管理に頼らなくても，あらかじめ管理するメ

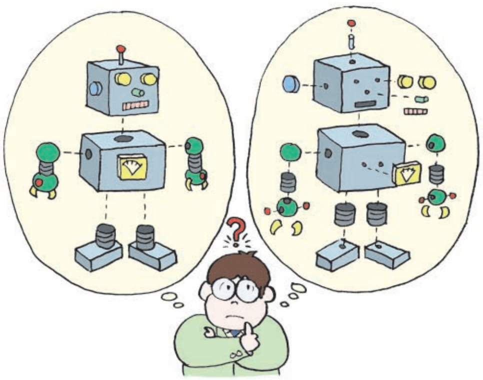


図10　構成要素をきめ細かくすると管理コストは増える

Column

環境をそっくり保管する最高の方法は？

環境をそっくり保管するのにいちばん手軽で確実な方法は，開発に使った環境をコンピュータごと梱包して倉庫に入れることです．もちろん開発に使っていたものですから確実に機能することは保証済みです（ただし保管費用はかさむ）．

それではいったい何を保管すればよいのでしょうか．マニュアル，メディア，ICE（in-circuit emulator）などの開発に使用した機器類など，すべての文書と環境を保管することになります．開発環境のみならず，文書の編集環境や印刷環境，すなわちアプリケーション・ソフトウェアやプリンタまで保管しなければなりません．

そこまで保管していても，まだ危険はあります．しまい込んだ製品の製品寿命です．樹脂や油脂，磁気メディアは劣化します．テープ・ドライブではゴムなどが溶けてしまうことがありました．このように，情報を納めることはできても，放置しておくと取り出せなくなる場合があります．光磁気ディスクは安定しているメディアとして知られており，約50年間の保管が保証されています．化学変化を利用するもの（CD-R，DVD-R）よりも物理変化によるもの（MOディスク；magneto optical disk）がお勧めです．文書は，紙（中性紙）での保管が必須です．

数年ごとに記録媒体の見直しを行い，新しい媒体に移し替えなければなりません．今，8インチ・フロッピ・ディスクがあったとして，手軽に参照できるでしょうか．

カニズムがあるのであれば，それを採用するべきです．そして，そのメカニズムに連動するように構成管理のインターフェースを定めます．

このほか，どの領域まで構成管理の対象に含めるのかを設定する必要があります．製品の企画から開発，サポート，廃棄に至るまで，一気通貫で構成管理を実施できれば理想的です．最小領域は個人作業ですが，最大領域はどのあたりでしょうか？ SCM（supply chain management），CRM（customer relationship management）までいくのでしょうか？ その限界は，各プロジェクトの状況によってさまざまでしょう．

さらに，製品開発の各段階において，どの情報が重要なのかは変わります．例えば，開発ではモデル情報が重要となります．一方，サポート・サービスでは個体識別情報が重要となります．

● 構成管理ツールを利用する

構成要素を登録し，関連づけを行うためには，何らかのツール（市販ツール，フリー・ソフトウェア，内製ツールを含む）が必要です．もし，だれも構成管理について実施したことがないという環境であれば，手もとのテキスト・エディタや表計算ソフトウェアを利用するところから始めてもよいでしょう．ただし，テキスト・エディタや表計算ソフトウェアでは，複数の人たちの間で共有し，更新するには難点があります．

市販のツールは，すでにいくつもの適用例が得られているため，ある程度，設定作業の省力化が期待できるかもしれません．ただし，道具（ツール）一般について言えることですが，適用する対象によって道具を取捨選択しなければなりません．万能の道具というものはあり得ないのです．



図11　作業者にフィードバックをかけること！
作業者が積極的に情報を入力するように，かならずフィードバックがかかるしくみを入れる．

また，道具を導入することが目的になってはなりません．目的は，あくまでも「もの」の構成を管理し，ほかの管理に使えるように最新の状態を維持することにあります．

「何をどのように管理したいのか」という構想を固めてから道具の選択を行うべきです．そして，可能であれば，道具に合わせて作業を見直してもよいでしょう．

ツールを使わせたい場合は，「そのツール経由でなければ構成要素にアクセスできない」ように設定します．

● フィードバックが運用のかぎ

管理者は，作業者がどのような手順に従って作業すればよいのか，というしごとの流れを定義しておかなければなりません．この流れの中で，かならず作業者にフィードバックがかかるしくみを入れましょう（**図11**）．例えば，構成管理情報を入力しないと製品を構成できないようなしくみを整えれば，つねに最新の構成管理情報を維持できます．これは直接的に強制する一つの方法です．一方，管理者が「構成管理情報を参照してしごとをしたかしなかったかを判断し，給与を査定する」と発表したらどうなるのでしょうか？ 作業者はとても積極的に構成管理情報を更新してくれることでしょう．

構成管理に携わる専任者を置くのと置かないのとでは，その後の運用に大きな違いが生まれます．専任者を置けば，最新の情報が維持されるように手配させることができます．もちろん，専任者は管理者に現状を報告する必要があります．

● 終わりこそきちんと管理する

プロジェクトには，いつか終わりが来ます（継続し続けるものはプロジェクトとは呼ばない）．プロジェクトが終わるとき，管理要素をうまくまとめるためには何を重視するべきなのでしょうか？

このときに思い出していただきたいのが，前項で述べた「作っているときと保守するときの価値観は異なる」という事実です．プロジェクトを終えるときには，製品を作るときに必要な要素ではなく，保守や再利用に必要な要素を管理するように目的を絞り込みます．

こうしてプロジェクトをうまくまとめることができれば，そこを足がかりにして，さらに先に進むことができるでしょう．

2 ここから始めるソフトウェア構成管理

ここでは，小規模な開発に携わっており，今後，構成管理を始めようと考えておられる読者の方に，身の回りにあるツールで実現できる構成管理を提案します．

● まずはバージョン管理から

「構成管理を行っていますか？」と聞くと「はい．CVS (Concurrent Versions System)を使っています」という回答がよく返ってきます．ところでCVSとは，オープン・ソースのバージョン(版数)管理ツールです注2．前節を読んだ読者の方は何がおかしいのかわかりますよね？ 構成管理とは，バージョン管理のことではありません．

しかし，バージョン管理はソフトウェア(あるいはHDLコードなど)の構成管理には必須のものです．開発の途中で前のバージョンの情報が役立つことが多いからです(ソース・コードを過去のある時点のものに戻したい，前のバージョンとの違いをチェックしたい，現在のバージョンでは削除してしまったけれど前のバージョンで実装していた機能をどう実現していたかを確認したい，など)．それなのに，ソース・コードを保管するためにバージョン管理ツールを使わず，作業領域のフォルダ(あるいはディレクトリ)を丸ごと別の場所にコピーしておくだけの開発者が意外と多いようです．たぶん読者の中にも，今ドキッとされた方がいるはずです．

● バージョン管理ツールを使おう

バージョン管理ツールを使わずにソフトウェアを開発している読者の方に質問します．こんな経験をしたことはありませんか？

1) ソース・コードにちょっとした変更を加えたら突然動作がおかしくなった．元に戻して変更し直そうと思ったが，どこを戻せばよいのかわからなくなってしまった．

2) 作業領域には，xxx.c, xxx2.c, xxx3.c, xxx.c.back, xxx.c.origといった，よく似たソース・ファイルがある．

3) 昨日行ったソース・コードの変更内容が思い出せない．

4) 古いデータやプログラムを使ってROMデータを出してしまい，マスクROM代をむだにした．

5) 開発したプログラムに不ぐあいが発生したが，その不ぐあいのあったプログラムに使ったソース・コードがはっきりしない．

こういった問題は，バージョン管理ツールを使うことでかなり解決できます．なぜなら，「今，このときの状態に戻りたいかもしれない」という状態をバージョン管理ツールに登録しておけば，いつでもその状態に戻れるからです．また，登録した状態と現在の状態の差分もかんたんに確認できます(**図12**)．変更を登録するとき，いっしょにコメントを登録できるので，どのように変更したかという記録も残せます．良いことずくめです．使わない手はありませんよね．しかも，使いかたも難しくありません．

コーディング段階になって，1ヵ月間コーディングだけして，それからまとめてテストするという人はいません．こつこつと日々コードを書き，ひとまとまりの機能を実現した段階でテストして，思うように機能が実現しているかどうかを確認します．うまく確認できれば，ソフトウェアを作る人はホッとできます．このときにバージョン管理ツールに登録すると，この幸せな状態にいつでも戻ることができます．幸せを貯金するようなもので，日々の楽しみと思えるようになればしめたものです(**図13**)．

こう考えると，バージョン管理ツールはソフトウェア開

```
[~/work] sennin% cvs checkout -d working cq
cvs checkout: Updating working
U working/sample.c
[~/work] sennin% cd working
[~/work/working] sennin% vi sample.c
[~/work/working] sennin% cvs commit -m "change newline
                                          code" sample.c
Checking in sample.c;
/Users/sennin/cvsroot/cq/sample.c,v  <--  sample.c
new revision: 1.2; previous revision: 1.1
done
[~/work/working] sennin% vi sample.c
[~/work/working] sennin% cvs diff sample.c
Index: sample.c
==================================== =======
RCS file: /Users/sennin/cvsroot/cq/sample.c,v
retrieving revision 1.2
diff -r1.2 sample.c
9a11
>                        /* fall through */
13c15
<                defoult:
---
>                default:
[~/work/working] sennin%
```

変更を確定するコマンド
古いバージョン(1.1)と新しいバージョン(1.2)の差分を確認するコマンド
追加された内容を表示している
変更された内容の変更前と変更後を表示している

図12　バージョン管理ツールで差分情報を確認しているところ
この例ではCVSを利用している．

注2：CVSはGNUソフトウェアとして提供されている．
詳細はURL「https://www.cvshome.org/」などを参照のこと．

発の必須アイテムと言っても過言ではありません．バージョン管理ツールを使って幸福を貯金する良い習慣をぜひ身につけてください．そして，この習慣が，構成管理の第1歩になります．

● コメントを利用して改訂内容を記録する

さて，プログラムを修正するのはどういうときでしょうか？ 例えば，以下のように大別できます．

1) プログラム仕様は理解していたが，プログラムの実装に問題があった場合

2) プログラム仕様の理解が不十分で，不十分な理解のもとにプログラムを作っていた場合

3) プログラムの処理性能や使用メモリ・サイズなどの非機能要件を満たすために実装方法を変える場合

4) プログラム仕様の記述があいまいだったりまちがっていたりして，不十分な仕様に基づいてプログラムを作っていた場合

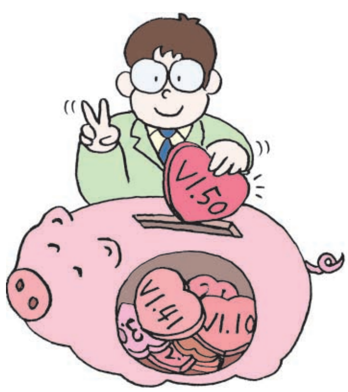


図13　プログラマは幸せを貯金できる

5) プログラム仕様に変更があって，プログラムを変更する場合

1) や **2)** は，いわゆるバグがあった場合です．プログラムを改訂して，修正が正しいかどうかを確認し，それからバージョン管理ツールに登録します．そのときにどんなバグ改修を行ったかをコメントとして入れることを忘れないようにしましょう．不ぐあい報告書をどの段階から作成するかは会社やプロジェクトによって異なるでしょうが，きっと不ぐあい報告書を作成するでしょうから，その不ぐあい報告書の番号をバージョン管理ツールのコメントに入れておくと，あとで改修内容を把握するのに便利です．筆者の会社では，開発途上で発生した障害の内容，処置方法，改修状況をデータベースとして管理する障害管理ツールを自作しているため，その番号で管理しています．

3) は，なんらかの観点から改良した場合です．改良内容をバージョン管理ツールのコメントに入れることはもちろん，改良のためのアイデアや方法をメモに残し，それにも通し番号を付けて，その番号をコメントに入れておけば，これも便利です．

4) は，プログラムの更新という意味では **1)** と同じですが，プログラムだけでなく，該当する仕様書も改訂する必要があります．**5)** も同様で，プログラムと仕様書の両方を改訂する必要があります．このような場合，当然のことながら仕様書を改訂してからその内容を確認し，プログラムを改訂します．

このとき，仕様書改訂が生じたことをソース・コード上にも記録として残しておく必要があります．一つの方法は，フ

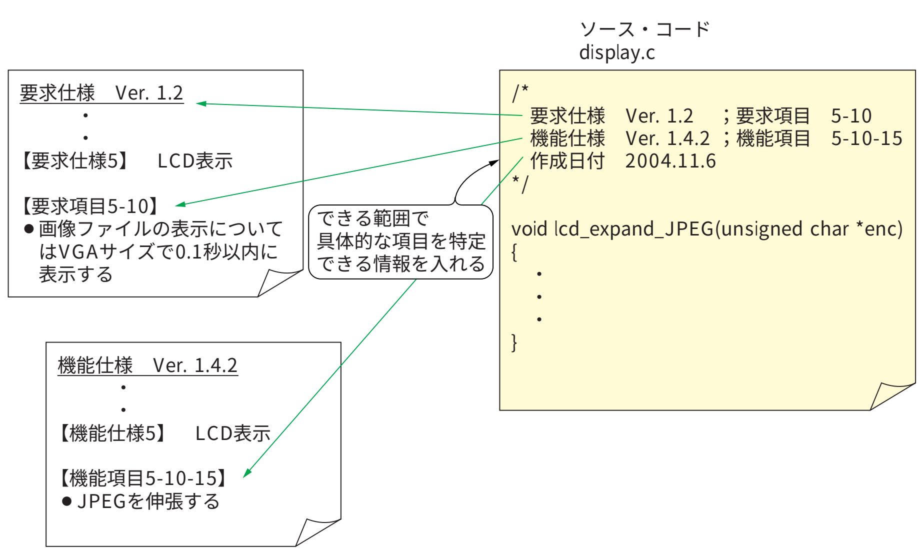


図14
ソース・コードと仕様書の連携
ソース・コードの中に，要求仕様のどの項目に対応しているのかをコメントとして記述しておくと，後でわかりやすい．

ァイルや関数の先頭に，どの仕様書のどのバージョンに基づいてプログラムを作っているかを残すことです（**図14**）．できれば，その仕様書のどの部分（例えば「3.4章」など）に基づいてプログラムを作成しているのかを書いておけばもっと便利です．これにより，仕様書のどの変更によってプログラムを改訂したのかがわかります．もちろん，そのためには仕様書もバージョン管理を行わないといけません．仕様書のようなデータもバージョン管理ツールを使って管理可能です．ただし，テキスト・ファイルの場合にはバージョン管理ツールは前のテキストとの差分として記録してくれるので効率的に保存できますが，バイナリ・データでは差分という形を取れず，効率的に保存できません．そのため，筆者の会社では，見やすさを重視してHTMLファイルで管理し，仕様書へのハイパー・リンクと関係者に読んでもらいたいコメントを合わせて関係者に公開しています（**図15**）．

あとは，プログラムと対応づける仕様書をどのように作成するのかを考慮する必要があります．例えば，「ソース・ファイル1は要求仕様書Ver. 1.2に基づいて作成した」ことがわかっても，仕様書が分厚ければ，具体的にどの部分かはわかりにくいものです．このとき，**図14**に示したように「要求仕様Ver. 1.2の要求項目5-10に基づいて作成した」ことがコメントとして書かれていると，ぐっとわかりやすくなります．

また，機能を記述している仕様書でも，ひんぱんに仕様が変わる箇所（外部とインターフェースを取る部分など）とそれほど変わらない箇所（核となる計算処理やアルゴリズム）があります．これらの部分を実現しているプログラムをいっしょに管理すると，仕様書がひんぱんに変更され，変更のたびに全体としてテストする必要が生じるので，効率的ではありません．このような場合，**図16**に示すように分割して管理することも一考に値します．これについては，開発当初からそのように判断して分割する場合もあれば，改訂，保守を重ねるに従ってわかってくる場合もあるでしょう．

● ところで仕様書，書いてますか？

さて，以上のことからわかるように，構成管理を行おうと思うと，プログラムと仕様書をきっちり対応づける必要があります．仕様書を書く，このあたりまえのことができていないプロジェクトが意外と多いようです．筆者が社外で講義するときに受講者に質問してみても，仕様書を作成しないでいきなりプログラムを書いているところがあるようです．また，仕様書はいちおう作成するけれど，開発が始まると，開発内容をいちいち反映させることができずにいるところも多いようです．単に「仕様書を書け」と言われてもなかなか書けないのが現状です．仕様書とプログラム開発を結び付けて開発を進めるたいせつさや有効性を理解して，「仕様書を書かなければ…」とみずから思うことが重要です．

構成管理を考えるということは，単にツールを使うことではなく，どういう手順と約束でソフトウェアを開発するかというところに関係してきます．したがって，まずは興味や関心を持った人が自分のプロジェクトで試しながら，周

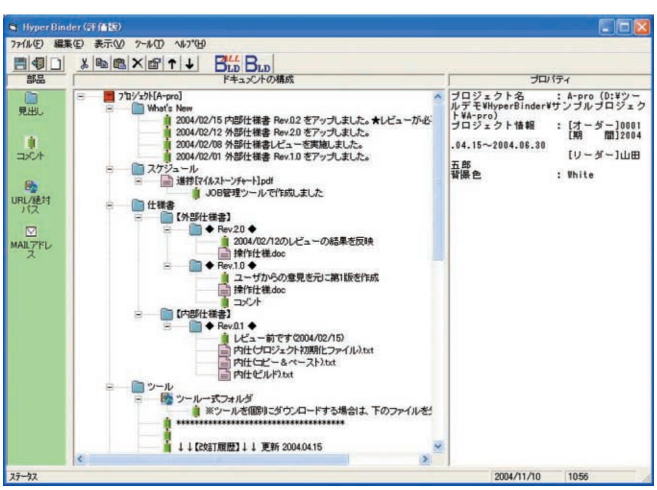


図15　仕様書の管理例（Webブラウザを利用）
なお，HTMLファイルへの記入作業を楽にするため，表示用HTMLファイルを自動生成できる入力用ツールを自作した．

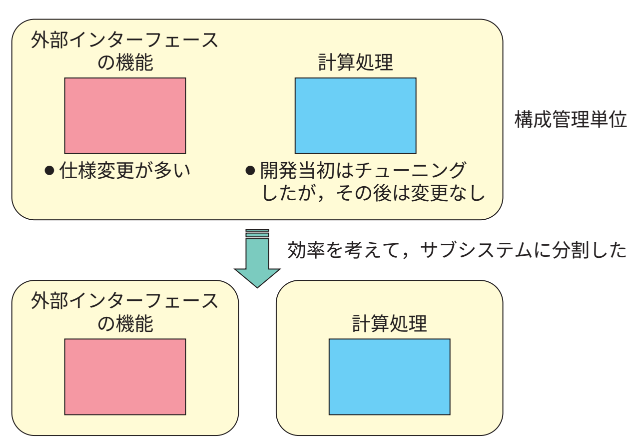


図16　仕様書と構成管理
構成管理単位を分割したほうが効率的な場合もある．

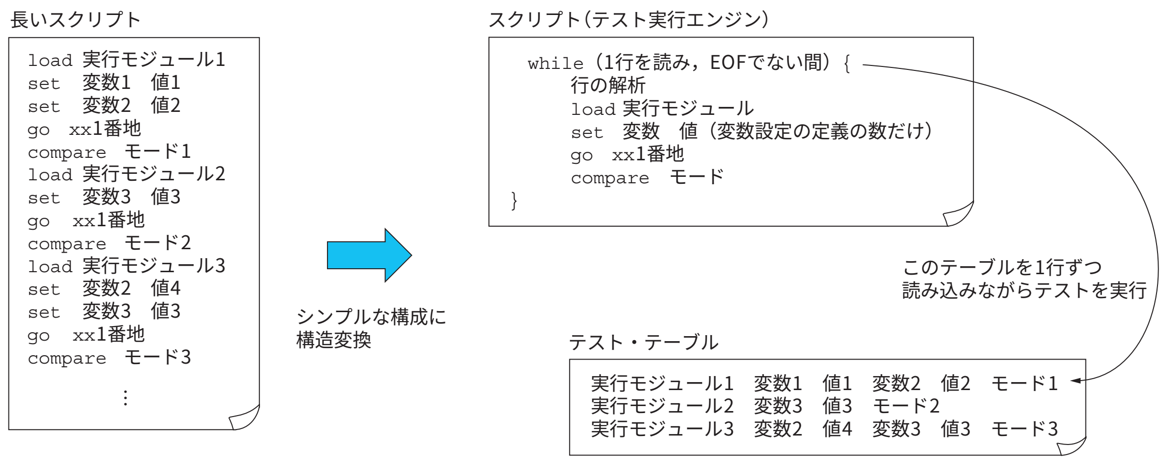


図17 テーブル・ドリブン型のテスト実行
テストの手順(スクリプト記述)とテスト項目(テーブル)を分けているため,見通しが良く管理しやすいテストになっている.

りのソフトウェア開発を見つめ直していく必要があります.

● テスト環境も忘れずに管理しよう

構成管理は,ソース・コードのバージョン管理だけでなく,仕様書や不ぐあい情報とも連携して管理する必要があることを説明しました.ところで,仕様書や不ぐあい情報だけでなく,テスト環境の管理もぜひ忘れないでいただきたいものです.

テスト環境はソース・コードと同等,いやそれ以上に重要なものです.ソース・コードあってのテストですが,テスト・パターン作成と期待値の作成,テスト実行環境の構築,データ変換や実行結果の期待値を比較するためのツール類など,テスト環境の構築にはソース・コードの開発以上に時間をかけているはずです.そのため,ソース・コードの構成管理を行うのなら,当然これらのテスト環境の構成管理も実施してください.

テスト環境の構成管理を行うのなら,テスト環境の変更内容が見やすくなっているほうが便利です.例えば,以下のようなことがわかるしくみになっていると効率的です.

1) テストの手順は前のバージョンとどう違うのか
2) どんなテスト項目が増えたのか(減ったのか)

くふうのしかたはいろいろ考えられます.それぞれの現場にあった方法でくふうしてください.

ここではくふうの一例として,本誌2004年9月号,pp.46-48の「シンプルなテスト・スクリプトで信頼性と生産性を上げる」[(2)]で解説したテスト実行用のスクリプトを挙げます.テストのためのスクリプトも,長い複雑なものになるとスクリプトそのものをテストする必要性が出てきてしまうので,スクリプトはできるだけ短くし,テスト項目は別ファイル(1回のテストに必要な情報を1行に書き,テストの数だけ行数がある)として用意しました.テスト・スクリプトはそのファイルを1行ずつ読みながらテストを実施するという,テーブル・ドリブン型のテストです(**図17**).この形式だと,**1)**の「テストの手順」がスクリプトの内容に,**2)**の「テスト項目」の増減がテーブルの行の増減にあたります.テーブル・ファイルはテキスト形式なので,バージョン管理ツールの機能を使えば前のバージョンとの差分がわかります.さらに,スクリプトもバージョン管理ツールを使って登録し,その際にコメントとして追加・削減したテスト項目番号を入力しておけば,前のバージョンとのテスト手順の違いもわかりやすくなります.

● レビューはたいせつな儀式

いろいろとくふうして構成管理を行ってきた開発も,どこかのポイントで内容をチェックする必要があります.開発チーム内でレビューしてもよいですし,開発チーム外から第三者を迎えるのもよいでしょう.ひとりだけの開発なら,気持ちを切り替えながらやってもいいと思います.肝心なことは,各種資料がそろっているか,更新されているか,仕様変更がまちがいなくソース・コードに反映されているかを確認することです.閻魔(えんま)大王に日ごろの行いを聞かれるようなものです(**図18**).

このとき,日ごろから仕様書やソース・コード,不ぐあい報告書などを更新するたびにバージョン管理ツールに入



図18　日ごろの行いを反省するもまた良し

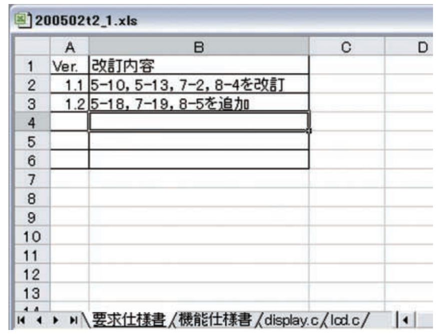


図19　開発成果をチェックする
各工程の変更内容をチェックし，改訂漏れがないかどうかをチェックする．

力してきたコメントが役立ちます．各ドキュメントごとにコメント一覧を抽出して，それぞれを1枚の表（Excelのシートなど）にまとめます（**図19**）．これをもとに開発を振り返りながら，各種変更がきちんと反映されているかどうかをチェックします．やりかたの一例としては，まず設計工程の上流のほう（要求仕様や機能仕様など）から変更内容をチェックし，それが下流工程に反映されているかどうかをチェックします．それから，今度はソース・コードの改訂記録を確認し，改訂になった原因が仕様書の記述に起因するものであれば，仕様書も同様に改訂されているかどうかをチェックします．

チェックが済めば次の段階に進みます．これが開発段階の途中のチェックなら，これでここまでの開発が検証できたので，この時点の仕様書やソース・コードを今後の開発のベースとします．これが出荷前のチェックなら，そのままソフトウェア構成監査になり，いよいよ出荷です．

● 構成管理は心がけから

今後，構成管理を始めようと考えられている方を想定して，どのようなことを考慮して進めていかなければならないかを説明しました．ここでは，構成管理を実践するにはどのようなことを考慮に入れてプログラムを開発しなければならないかをわかりやすく書いたつもりです．この説明を参考にくふうしていただければと思っています．構成管理に関する書籍もいろいろとあるので，詳細はそちらを読んで勉強してください．

また，ここでは小規模な開発に携わっていても構成管理は絶対に必要な技術であり，身の回りにあるツールを用いてできるというつもりで書きました．最近は構成管理のための便利なツールもありますが，まずはツールありきではなく，構成管理の概念を理解し，身の回りにあるツールを用いて運用し，自分の職場に合った適用をくふうしてみてください．そうでないと，効果的にツールを活用することはできないでしょう．

参考・引用*文献

(1) 居駒幹夫，横田一樹，田中友子，酒井由夫，平野誠太郎，北村裕之，斎藤賢一，大西健児，石井雄一；4th Open SESSAME Seminar 組込みソフトウェア技術者・管理者向けセミナー 中級者向けテキスト2，2004年9月．
(2) SESSAMEの仙人；シンプルなテスト・スクリプトで信頼性と生産性を上げる，Design Wave Magazine，2004年9月号，pp.46-48.

やまさき・たつお
せさみ・の・せんにん
組込みソフトウェア管理者・技術者育成研究会（SESSAME）

1

<筆者プロフィール>

山崎辰雄．別名，SESSAMEの恐竜．プロセッサ開発からOS開発，ERP(enterprise resource planning)コンサルティングまで，ハードウェアでもソフトウェアでも「おもしろければ何でもこなす」技術系遊び人．
SESSAMEの仙人．マイコン向けミドルウェアの企画・開発と社内のソフトウェア開発支援に従事している．最近は組み込みソフトウェア開発の方法論や技術者教育に関心を持っている．「おもしろいことを自分なりに」をこだわりにする京都派エンジニア．